Attorney Docket No. 82937

FUNCTIONAL ELEMENT TEST TOOL AND METHOD

STATEMENT OF THE GOVERNMENT INTEREST

The invention described herein may be manufactured and used by or for the Government of the United States of America for Governmental purposes without the payment of any royalties thereon or therefore.

CROSS-REFERENCE TO RELATED PATENT APPLICATIONS

Not applicable.

BACKGROUND OF THE INVENTION

(1) Field of the Invention

The present invention relates generally to software testing and, more specifically, to a software test tool operable for providing an independent test environment for testing computer program system functional elements which may have multiple Application Data Exchange (ADE) interfaces.

Description of the Prior Art

Large scale hardware and software systems and application may typically include many different functional elements or modules. In many cases, the various functional elements or modules are developed concurrently. As used herein, a functional element is a software module which performs a unique software task and which may have multiple interfaces with other functional elements and/or with an application comprised of numerous functional elements and/or with an overall system comprised of a plurality of applications. In a preferred embodiment of the present invention, the functional element performs one or more tasks which utilize an inter-task interface or module-to-module communication protocol or mechanism. Each functional element may have multiple interfaces. The interface sets forth constraints on formats, timing, and/or other factors required by an interaction of functional elements that perform different tasks within a computer system.

Once the system is assembled, various problems may occur that cannot be easily traced to a particular functional element or module. In many cases, the problems relate to errors that occur at the functional element or module interfaces. As an example, one module or functional element might test and analyze

different underwater acoustic propagation signal propagation loss models. If the overall system response is not as expected, it may be difficult to determine whether the particular functional element for analyzing signal propagation loss is operating correctly or whether the fault lies elsewhere. Moreover, it may typically be difficult to monitor software interface operation of any particular functional element with respect to other functional elements and the overall system.

Where systems involve complex meshings of inter-process communications, a known technique for implementing the communication of task messages has been to use a shared memory for passing data between tasks. However, with this technique the problem of determining the fault which produces a given error is even greater.

Consequently, there remains a long felt but unsolved need for improved testing of functional elements to ensure that accurate processing occurs within each functional element and that communications between the functional elements are consistent with the interface protocols. Those skilled in the art will appreciate the present invention that addresses the above and other problems.

SUMMARY OF THE INVENTION

Accordingly, it is an object of the present invention to provide an improved system and method for testing functional elements of a computer software system.

Another object is to provide a system and method as aforesaid which provides a controlled test environment that can facilitate off-line quantitative analysis of the test results and collected data.

A further object is to provide a system and method as aforesaid which is a stand alone functional test tool that is much easier and faster to operate than a debugger or hand inspection of the application output results.

A still further object is to provide a system and method as aforesaid which enables software to reduce the overall number of defects that occur during the development phase.

A yet further object is to provide a system and method as aforesaid which is of special utility in application systems in which inter-process communication is implemented by techniques of using a shared memory scheme in order to pass data between tasks.

These and other objects, features, and advantages of the present invention will become apparent from the drawings, the

descriptions given herein, and the appended claims. However, it will be understood that above listed objects and advantages of the invention are intended only as an aid in understanding aspects of the invention, are not intended to limit the invention in any way, and do not form a comprehensive list of objects, features, and advantages.

In accordance with the present invention, a method is disclosed for a stand-alone testing environment for a functional element of an application. The application may be a subpart of an overall system. The functional element may have a plurality of interfaces with the application and/or overall system. The method may comprise one or more steps such as, for instance, providing a computerized dialog to permit a user to create an input data file for the functional element, prompting a user for functional element interface tasks that have been previously developed utilizing the stand-alone testing environment. Alternatively, the dialog permits a user to start the functional interface tasks and a user supplied application system task. The plurality of interfaces are monitored.

Other steps may include storing a unique interface file for the functional element interface tasks and/or displaying a status window while running the functional interface tasks. In

a preferred embodiment, the step of creating a test case generation file provides the user with task creation options related to the functional interface tasks. Other steps may include compiling the test case generation file to create a test case executable file for producing the functional interface tasks and/or storing the input data file in a user defined file such that the user defined file may be viewed and edited outside of the stand alone testing environment.

An embodiment of the present invention provides a method for testing a functional element of a computer system with a stand-alone functional element test tool wherein the functional element has at least one interface for communicating with other functional elements of the computer system using an interface protocol. In this embodiment, the method comprises one or more steps such as, for instance, creating an input data file for the functional element by prompting a user for data format and content, storing the input data file, creating a test generation file by providing the user with a plurality of task creation options such that the selected task creation options are input into the test generation file, compiling the test generation file to produce a test case executable file in a predetermined inter-process communication protocol based on the selected task

creation options, initiating a test utilizing the test case executable file and the input data file for testing the functional element and the interface(s) by monitoring a status of the test, and storing test result data related to the test.

The step of creating a test generation file may further comprise selecting test initiation features and/or providing at least one user defined button wherein the user defined button is user operable for the step of initiating the test. The method may further comprise playing back the test result data and/or providing a file viewer for the input data. Thus, the method permits comparing the test result data with expected results from the functional element utilizing the input data file.

Thus, the present invention also provides a system operative for testing performance validity and accuracy of a first computer program functional element which may comprise one or more elements such as, for instance, a test case data file producing subsystem for facilitating the production by a user of at least one file of test case data. The test case data producing subsystem can then be used for identification of an input data structure in order to prompt a user for input values of the test case data. Preferably, the test case data producing subsystem is operative to store one or more files of test case

data. Other elements may include a test case generation file producing subsystem for facilitating the production by a user of a test case generation file. The test case generation file producing subsystem provides a plurality of user options to facilitate the user in testing operation of the first computer program functional element and at least one communication interface. Moreover, the system may comprise a test case execution subsystem to operate the first functional element based on the selected user options and the test case data. The test case execution subsystem preferably provides a monitor for operation of the first functional element. Preferably the test case execution subsystem is operable for executing operation of a compiled executable file produced from the test case generation file. The test case execution subsystem may preferably be operable to effect operation of a second functional element simultaneously with operation of the first functional element such that the test case execution subsystem is operable to monitor the interface between the first function element and the second functional element. The test case execution producing subsystem is also preferably operative to tag an output of the first functional element with an indication of the task status. The test case generation file producing

subsystem is also preferably operative to facilitate the user selecting a predetermined test initiation event to start flow of the test case data into the first functional element.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the invention and many of the attendant advantages thereto will be readily appreciated as the same becomes better understood by reference to the following detailed description when considered in conjunction with the accompanying drawings wherein corresponding reference characters indicate corresponding parts throughout several views of the drawings and wherein:

FIG. 1 is a schematic which indicates the general functioning of a stand-alone test tool in accord with the present invention;

FIG. 2 is a schematic of an embodiment of a stand-alone test tool as the tool may be configured to cooperate with a given computer application system; and

FIG. 3 is a block diagram indicating a flow and structure of a stand-alone test tool in accord with the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring to FIG. 1, in accord with the present invention, a stand-alone functional element (SAFE) test tool 10 has been developed to provide an independent test environment for one or more functional elements 18, which are desired to be the object of testing, within a given application or system. In a preferred embodiment, the SAFE test tool can be utilized to test any test object software which utilizes an inter-module communication mechanism or interface protocol.

An illustrative embodiment of the inter-process communication protocol is the Application Data Exchange (ADE) protocol, which has been custom developed for the Navy for use in connection with software application programs. Illustrative of one such Navy application program is a Sensor Performance Prediction Functional Segment (SPPFS) large-scale software module. SPPFS has multiple interfaces with software programs of an integrated system of ASW equipment digital processors. Briefly, SPPFS performs casting and analysis of different acoustic signal propagation loss models used in ASW combat control. More particularly, it functions in connection with sonar-based antisubmarine warfare equipment known as system AN/SQQ-89(V) 15. However, both the SPPFS and the AN/SQQ-89(V)15

are referred to for purposes of illustrating an environment, in which the present invention is utilized, and in and of themselves form no part of the invention.

ADE uses queues to communicate between application tasks. The concept of queue within ADE can be viewed as a mechanism for inter-process signaling. Each queue consists of one or more messages. Messages in turn contain information. An example of a queue with an implementation in the environment of ASW systems is the queue which is used by a propagation loss analysis function task element of a sonar signal analysis computer program to send equipment settings to the program's interface function task elements. This provides communication between the two tasks.

ADE uses a shared memory scheme in order to pass data between tasks. ADE manages the shared memory. Therefore when employed with SPPFS this functionality is invisible to the implementation of the SPPFS program task. However, the memory requirements of the implementation must be known and allocated via ADE functions (e.g., AdeAlloc). Message passing with ADE can be viewed as the notification by an ADE protocol message to a task that information (sometimes in this specification and its

appended claims simply referred to as "data") is ready and that the data is located at a specific shared memory location.

ADE uses a hierarchical structure for storing information within the ADE database. There are four levels within the hierarchy, "Queue", "Application", "Topic", and "Item", where Queue is the most general and Item is the most specific. A Queue represents a highest level within the hierarchy. Within the SPPFS application program implementation, the Queue is typically the interface with the high level task. The Application is the second level and is associated with a queue. The functionality of an Application is typically more specific than that of a queue. However, within the SPPFS application the two often encompass the same functionality. The third level is a Topic which is associated with an Application. Again, the Topic is more specific than the queue. Finally the fourth level is the Item which is based on a Topic.

The SAFE test tool provides an excellent test bed for testing functional element changes as well as software drops because the user has the capability of generating, collecting, and responding to all interfaces of the specified system functional element. Although the SAFE test tool prompts the user for input to perform testing, it is assumed that the user

understands the test object functional element operation within the overall system as well as the inter-module communication or interface protocol, e.g., the ADE protocol.

Referring again to FIG. 1, there is schematically shown SAFE test tool 10 interconnected with any number of interfaces, as indicated by 12, 14, and 16, of test object functional element 18 to provide an independent test environment. Each of interfaces 12, 14, and 16 may have several modes of operation which may be tested in accord with the present invention. SAFE test tool 10 provides a stand-alone test environment that may be utilized to simulate the application or overall system 19 with respect to test object functional element 18 and monitor any number of interfaces for proper operation, such as element interfaces 12, 14, and 16. The SAFE test tool 10 and the application or system 19 each communicate via a suitable linking arrangement, such as computer bus 19-1, with the above-discussed shared memory 19-2, that is involved in the shared memory scheme to pass data between tasks. Also per this earlier discussion, shared memory 19-2 has four levels of hierarchical structure: Queue 19-2A, Application 19-2B, Topic 19-2C, and Item 19-2D.

SAFE test tool 10 can also be utilized in other configurations. For instance, in FIG. 2, SAFE test tool 10 may

be connected to one or more test object functional elements, such as functional elements 1, functional element 2, functional element 3, and functional element N, designated as 20, 22, 23, and 24, respectively, wherein communication paths 26, 28, 29, and 30, may comprise one or more ADE interfaces. Each ADE interface may have several modes of operation. In this embodiment, the test object functional elements may be part of or form a software application program such as, for example, one of the functional elements of the above described Sensor Performance Prediction Functional Segment (SPPFS) 31. In turn, SPPFS is part of an overall large-scale integrated system, such as the above referred to AN/SQQ-89(V)15 combat control system (not shown). The functional elements of SPPFS interface with various application programs of the large scale integrated system through multiple interfaces, with interface tasks constituting inter-process communications. There may be other elements such as a CORBA (common object request broker architecture) link 31-1 and a CORBA software bridge 32 which may be utilized with many functional elements utilizing, preferably, ADE interfaces. In another arrangement, AN/SQQ-89(V)15 External Interfaces 34 as indicated in dashed lines, may operate through

link 31-1 and CORBA bridge 32 for testing one or more functional elements 20-24 and the various associated interfaces.

FIG. 3 provides an overview of the details of operation of SAFE test tool 10. In order to effectively interface with a test object functional element, i.e., the functional element involved in the run test case step 86 of a test. SAFE test tool 10 preferably operates in several selectable modes of operation or, in other words, is comprised of several selectable subsystems, as indicated at select subsystem 50. These subsystems of SAFE test tool 10 permit the user to create input data 52, create an interface test 54, application, and to run the tests 56.

Input file creator mode or subsystem 52 of SAFE test tool 10 provides a mechanism that allows a user to create one or more input data files that can eventually be accessed during test run time. During input file creation, SAFE test tool 10 provides a dialog with the user as indicated at 58 and 60. SAFE test tool 10 allows the user to identify the structure of the input data and then uses this structure to prompt the user as to possible input values 60 in conjunction with user dialog 58. This is accomplished by prompting the user for data format and content. The format may be manually entered or the format may be

automatically determined. For instance, utilizing a drop-down form or other selection means, the user may identify the data structure to be utilized. Once the data format is identified, the data format is then utilized by SAFE test tool 10 to prompt the user to enter input values. When completed, the data is stored in a user-defined data input file 62. User-defined data input file 62 can now be used as an input file for future test cases. Preferably, the format of user-defined data input file 62 is such that the file contents may be viewed and modified outside of the environment of SAFE test tool 10 for ease of use. For instance, the file may be readable utilizing file editor 64 which may be a text file editor or any other suitable editor which preferably does not require the use of SAFE test tool 10.

Create interface test case mode or subsystem 54 of SAFE 10 provides a mechanism that allows the user to create a test case generation file. In one illustrative embodiment, the test generation file is recorded in the form of a high-level interface language, custom developed for ADE. The development of such a custom language is within the routine skill of programmers who develop special purpose compilers. More generally, the high-level interface language can be a development for any other interface protocol where such other

interface protocol is involved. In this embodiment, it is assumed that the user is knowledgeable of the functional element interface task to be tested. The file generation is accomplished by providing the user with interface task creation options 66 and user input 68 and may include options such as Create Queue, Edit Queue, Begin Enumeration, and the like.

This description now proceeds to details of various preferred functional element task creation options. Create Queue allows SAFE test tool 10 to be able to create a Queue component of an ADE message which represents a function or functions to be performed for a test case. This component comprises the information (herein and the appended claims sometimes simply referred to as "data") stored in the Queue level of the ADE hierarchical shared memory. For each Queue message component, SAFE test tool 10 allows the creation of a user-defined Queue message component processing procedure or the use of a default Queue message component processing procedure. SAFE test tool 10 preferably has an edit queue option whereby SAFE test tool 10 allows editing of each queue. However, in a one preferred embodiment, once the Queue is created, SAFE test tool 10 is not able to delete the Queue message component. Another option is the Process Queue Messages option whereby SAFE

test tool 10 preferably is able to add messages that will be automatically handled within the test case. For this purpose, SAFE test tool 10 writes data to an output file specified by the user. SAFE test tool 10 allows for the processing of multiple message contents in the same Queue message component. SAFE test tool 10 does not allow the addition of messages until the Queue message component for the test case has been created. The option is also provided for creating an Application component of an ADE message comprising data stored in the Application level of the shared memory, whereby SAFE test tool 10 preferably permits creation of multiple Application message components for each test queue. SAFE test tool 10 permits Application message component creation only after a Queue message component has been successfully created. An Edit Application option permits SAFE test tool 10 to modify each Application message component. A Create Topic message component option permits SAFE tool 10 to similarly create multiple Topic message components for each test application. Topic message components are capable of being created only after an Application message component has been successfully created. Edit Topic permits modification of each Topic message component. Create/Edit Item allows SAFE test tool 10 to be able to create and/or edit multiple Items message

components. Note that Items message components are created only after a Topic message component has been successfully created. Enumerate on Application, Topic, or Queue, allows SAFE test tool 10 to be able to begin an enumeration on an application, topic, or item that has been defined in the task to be tested. SAFE test tool 10 automatically saves the server, Queue message component, and Topic message component identifiers upon successful enumeration. In addition, SAFE test tool 10 preferably provides the capability of allocating memory, requesting data, or requesting notice when the enumeration is satisfied. An Allocate/Free Memory option permits SAFE test tool 10 to be able to allocate and free memory and associate each memory block action with a given action (e.g., enumeration). A Send/Receive Data option permits SAFE test tool 10 to send and receive data to the task being tested. An Initiate Input Data option preferably permits SAFE test tool 10 to provide the capability of initiating input data by allowing the user to create, for instance, buttons or other indicators. Upon selection of the button, SAFE test tool 10 would then initiate data to the interface and functional element being tested. The input data will be accessed from a user defined input data file. The option to Insert Header Files permits SAFE

test tool 10 to provide the user with the ability to insert header files into the software as needed. This can preferably be performed automatically or manually.

The selected options are input into a generation file that is used to create the functional element test case. One of the options, as discussed above, permits the user to accept/initiate data from/to the function element for the interface (preferably ADE) task being tested. Data may preferably be initiated automatically upon the occurrence of some event (e.g., receipt of a message) or manually via a user-defined button as discussed above. Upon completion of the test case generation process, test case generation file 70 is available. In a preferred embodiment, test case generation file 70 is compiled outside of SAFE test tool 10 utilizing, for instance, by a compiler 72 which compiles the various options selected to create test case executable 74.

Test case execution 56 of SAFE test tool 10 provides a mechanism to run multiple test case executables when testing a functional element application task as indicated at 84 and 86. In a preferred embodiment, this testing employs a predetermined interface protocol for inter-process communications, e.g., the ADE protocol. Test case executable files 74 are therefore the

compiled executable version of the test case generation file created within the test case creation 54 mode. For each test case, SAFE test tool 10 preferably provides a monitoring window, as indicated at 78, that displays the current status of the given task. During test case generation stage 54, preferably print statements (for instance print statements) are automatically inserted that generate or print the status in monitoring window 78. In addition, the user define options (e.g., data initiation buttons) are displayed as dictated by each test case. The run test case step 86 is then performed under initiation by a start options function 84, using as inputs the user defined filed 62 and test case executable 74. Upon completion of the task execution, the results are output to the test case and may be stored in data storage 80 for future analysis or playback such as indicated at 82. SAFE test tool 10 is operable to maintain a unique interface file for each functional element task as prescribed by the user.

Reference is again made to FIG. 3, for an example of use of SAFE test tool 10 in connection with the SPPFS software application program (see reference character 31, FIG. 2) and the operation of SPFFS in examining and analyzing different acoustic signal propagation loss models. Further, in this example, it is

assumed that SAFE test tool 10 employs the ADE interface protocol for inter-process communications. Both the interface requirements and the specific input data requirements are well defined within the SPPFS application. Varying input data can be supplied and outputs can be collected to data files for off-line quantitative analysis utilizing playback 82, e.g., pointwise comparison with simulated/expected results. In addition, comparison of the different acoustic signal propagation models with different environments can be facilitated via SAFE test tool 10. When using SAFE test tool 10 in this way, the user would be prompted for the command line that initiates the selected SPPFS functional element to be the object of testing by SAFE test tool 10. In addition, the user would be prompted for the interface tasks as developed with interface task creation options 66. These tasks would be initiated via SAFE test tool 10 with test case executable 74. Upon successful initiation at 84, the user would have the ability to initiate inputs to the functional element to be tested. The responses from the test object functional element being tested during run test case step 86 would be maintained via SAFE test tool 10 for later comparison or output validation.

In accordance with the present invention, any application that utilizes a particular inter-module communication mechanism or interface such as the ADE protocol, or other interface protocols, will be significantly enhanced by the use an embodiment of SAFE test tool 10. SAFE test tool 10 facilitates functional level testing and evaluation of the application system (in the example, SPPFS) functional elements independently of the remainder of the system. In addition, SAFE test tool 10 provides a controlled test environment that can facilitate off-line quantitative analysis of the test results and collected data. Finally, SAFE test tool 10 provides a good environment for verifying software changes and status at a functional element level. This mechanism is therefore a valuable tool in the validation of software performance and processing accuracy. Quality software can be developed in less time and with fewer overall defects due to use of SAFE test tool 10 during the development phase. This affords significant cost savings both during the initial development phase as well as overall life cycle maintenance.

It will be appreciated by those skilled in the art that the invention can be implemented using a suitable programmed general purpose computer or special purpose hardware, with program

routines or logical circuit sets performing as processors. Such routines or logical circuit sets may also be referred to as processors or the like.

Therefore, it will be understood that many additional changes in the details, materials, steps and arrangement of parts, which have been herein described and illustrated in order to explain the nature of the invention, may be made by those skilled in the art within the principle and scope of the invention as expressed in the appended claims.